# *Module 8*
# *Functions and Classes*

Tamer Abdou, PhD

IND 830 - Python Programming for Data Science

Data Science Laboratory

Ryerson University



Ryerson University
The Chang School of Continuing Education

# Outline

1. Functions
2. Classes



Ryerson University
The Chang School of Continuing Education

# Functions

1 Functions
Defining Functions
Abstraction Mechanisms

2 Classes

# The Syntax of a Function

```
def <function name>(<parameter 1>, ..., <parameter n>):
    <body>
    return <function output>
<function name>(<variable 1>, ..., <variable n>)
```

- The syntax of a definition function contains a header and a body
- The header consists of the reserved word '**def**' followed by the name of the function, then a parenthesized list to declare the input parameters, then a colon '**:**' at the end of the header
- The body of a function should be indented with the exact same spacing for each line and can have as many statements as required
- The reserved word '**return**' is used if the function has an output
- A function can be called by its name followed by a set of variables

# *Example 8.1 (Script)*

```
1  def rectangleAP(length, width):
2      #Returns the area and the perimeter using
3      #the length and width of a rectangle
4      Area = length * width
5      Perimeter = (length + width) * 2
6      return [Area, Perimeter]
7
8  resultRec1 = rectangleAP(40, 5)
9  resultRec2 = rectangleAP(5, 40)
10
11 print(resultRec1 == resultRec2)
12 print(resultRec1 is resultRec2)
```

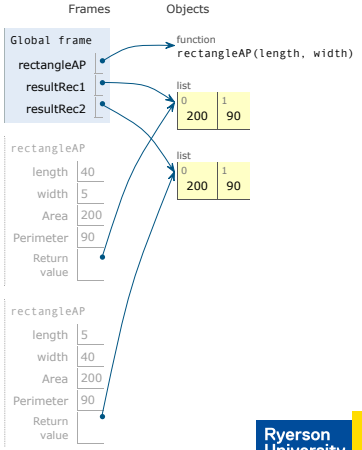
# Example 8.1 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  def rectangleAP(length, width):
2      #Returns the area and the perimeter using
3      #the length and width of a rectangle
4      Area = length * width
5      Perimeter = (length + width) * 2
6      return [Area, Perimeter]
7
8  resultRec1 = rectangleAP(40, 5)
9  resultRec2 = rectangleAP(5, 40)
10
11 print(resultRec1 == resultRec2)
12 print(resultRec1 is resultRec2)
```

Print output (drag lower right corner to resize)

```
True
False
```



# *The* `main` *Function*

- The `main` function is used to handle the user interaction with the program
- The `main` function is the designated entry point to the execution of the program
- A file's `__name__` is set to `__main__` when executed as a script
- If the file is imported as a module, the Python interpreter will set `__name__` to the module's name

# Example 8.2 (Script)

```
1  def main():
2      #The main function of the program
3      l,w = input("Enter length and width: ")
4      result = rectangleAP(float(l), float(w))
5      print("The area is", result[0])
6      print("The perimeter is", result[1])
7
8  def rectangleAP(length, width):
9      #Returns the area, perimeter
10     #of a rectangle using the length and width
11     Area = length * width
12     Perimeter = (length + width) * 2
13     return [Area, Perimeter]
14
15 if __name__ == "__main__":
16     # execute only if run as a script
17     main()
```

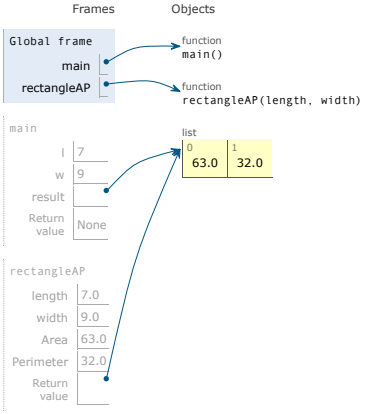
# Example 8.2 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  def main():
2      #The main function of the program
3      l,w = input("Enter length and width: ")
4      result = rectangleAP(float(l), float(w))
5      print("The area is", result[0])
6      print("The perimeter is", result[1])
7
8  def rectangleAP(length, width):
9      #Returns the area, perimeter
10     #of a rectangle using the length and width
11     Area = length * width
12     Perimeter = (length + width) * 2
13     return [Area, Perimeter]
14
15 if __name__ == "__main__":
16     # execute only if run as a script
17     main()
18
```

Print output (drag lower right corner to resize)

```
Enter length and width: 7,9
('The area is', 63.0)
('The perimeter is', 32.0)
```



# Abstraction Mechanisms

1. Eliminating Redundancy
2. Systematic Variations
3. Hiding Complexity

# Abstraction Mechanisms: Eliminating Redundancy

```
1  def rectangleAP(length, width):
2      #Returns the area and the perimeter using
3      #the length and width of a rectangle
4      Area = length + width
5      Perimeter = (length + width) * 2
6      return [Area, Perimeter]
7  print(rectangleAP(40, 5))
8  print(rectangleAP(10, 2))
9  print(rectangleAP(60, 6))
```

- Do not repeat yourself and create functions instead
- Code redundancy is not preferable:
  - It requires entering or copying the same code over and over (This is an inefficient practice)
  - Upon changing tiny parts, revising other related parts becomes compulsory (This can become a maintenance nightmare)


Ryerson University The Chang School of Continuing Education

# Abstraction Mechanisms: Systematic Variations

```
1  def rectangleAP(length, width):
2      #Returns the area, perimeter
3      # and ratio of a rectangle
4      #using the length and width
5      Area = length * width
6      Perimeter = (length + width) * 2
7      ratio = length/width
8      return [Area, Perimeter, ratio]
9  print(rectangleAP(40, 5))
10 print(rectangleAP(10, 2))
11 print(rectangleAP(60, 6))
```

- The function's code presents an algorithm in a general form to solve a class of problems
- By systematically varying either the problem instances, arguments, or results, a new algorithm can be generated without additional complexity


Ryerson University The Chang School of Continuing Education

# Abstraction Mechanisms: Hiding Complexity

- Hiding the calculations of relevant variables reduces the complexity of the system
- Also, hiding complicated steps and encapsulating them into one block helps the programmer focus on new objects without worrying about hidden details
- Functions can enforce a division of labour. Each function performs a single coherent task, such as computing an average of a set of numbers or calculating a perimeter of a polygon
- Each of the tasks required by a system can be assigned to a function, including managing or coordinating the use of other functions

# Classes

1 Functions

2 Classes
Objects and Classes
Docstrings, Accessors and Mutators
Polymorphism and Inheritance

# Objects and Classes

- Similar to functions, objects are abstractions
- Objects package a set of data values (object's state) and a set of operations (object's methods)
- A class is a blueprint for each of the objects of that class
- A class contains:
  - Definitions of all methods that the class's objects can recognize
  - Descriptions of all data structures used to maintain the state of the class's objects

# The Syntax of a Class

```
class <class name>(<parent class name>):
    <method definition 1>
    ...
    <method definition n>
```

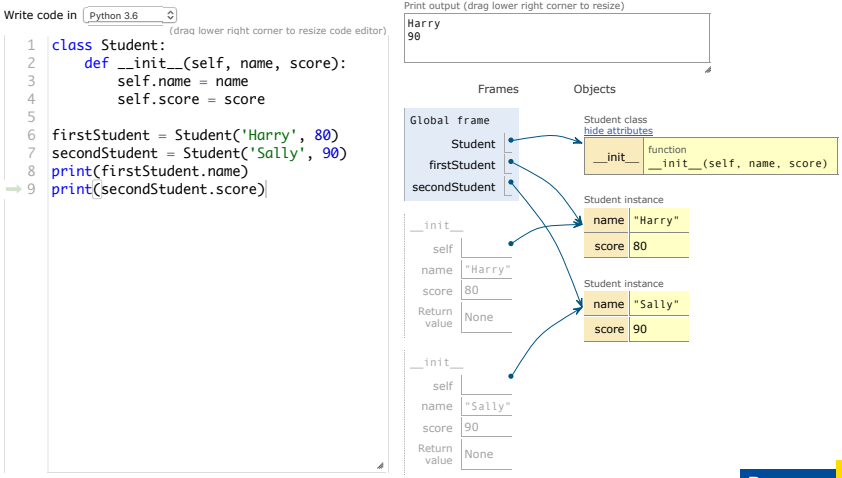
- The syntax of a class contains a header and a body
- The header consists of the reserved word '**class**' followed by the name of the class, then a parenthesized list to declare the parent-class name, then a colon '**:**' to end the header
- The body of a class should be indented with the exact same spacing and can have as many methods as required
- All Python classes, including the built-in ones, are organized in a tree-like **class hierarchy**


Ryerson University The Chang School of Continuing Education

# *Example 8.3 (Script)*

```
1  class Student:
2      def __init__(self, name, score):
3          self.name = name
4          self.score = score
5
6  firstStudent = Student('Harry', 80)
7  secondStudent = Student('Sally', 90)
8  print(firstStudent.name)
9  print(secondStudent.score)
```

# Example 8.3 (Program Execution)



# Docstrings, Accessors and Mutators

- **Docstrings**: Docstrings can occur either inside a module, class or method
- **Accessors**: Methods that allow a user to observe but not change the state of an object
- **Mutators**: Methods that allow a user to modify an object's state

# *Example 8.4 (Script)*

```
1  class Student:
2      def __init__(self, name, score):
3          '''For each student,
4          only two values
5          will be stored:
6          name and score '''
7          self.name = name
8          self.score = score
9
10 help(Student)
```


Ryerson University The Chang School of Continuing Education

# *Example 8.4 (Program Execution)*

Write code in Python 3.6

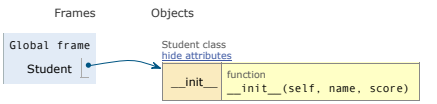
(drag lower right corner to resize code editor)

```
1  class Student:
2      def __init__(self, name, score):
3          '''For each student,
4          only two values
5          will be stored:
6          name and score. '''
7          self.name = name
8          self.score = score
9
10 help(Student)
```

Print output (drag lower right corner to resize)

```
Help on class Student in module __main__:

class Student
 |  Methods defined here:
 |
 |  __init__(self, name, score)
 |      For each student,
 |      only two values
 |      will be stored:
 |      name and score.
 |
```



# *Example 8.5 (Script)*

```
1  class Student:
2      def __init__(self, name, score):
3          self.name = name
4          self.score = score
5      def lastName(self, name):   #Accessor
6          nameParts = name.split(" ")
7          return nameParts[1]
8      def adjustedScore(self, score): #Mutator
9          self.score = score + 10
10         return self.score
11
12 firstStudent = Student('Harry Brown', 80)
13 print(firstStudent.name)
14 print(firstStudent.lastName(firstStudent.name))
15 print(firstStudent.score)
16 print(firstStudent.adjustedScore(firstStudent.score))
17 print(firstStudent.score)
```


Ryerson University The Chang School of Continuing Education

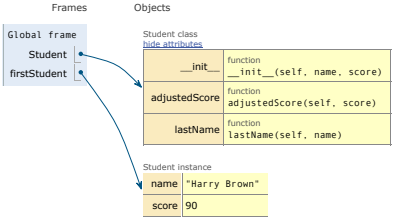
# Example 8.5 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  class Student:
2      def __init__(self, name, score):
3          self.name = name
4          self.score = score
5      def lastName(self, name):   #Accessor
6          nameParts = name.split(" ")
7          return nameParts[1]
8      def adjustedScore(self, score): #Mutator
9          self.score = score + 10
10         return self.score
11
12 firstStudent = Student('Harry Brown', 80)
13 print(firstStudent.name)
14 print(firstStudent.lastName(firstStudent.name))
15 print(firstStudent.score)
16 print(firstStudent.adjustedScore(firstStudent.score))
17 print(firstStudent.score)
```

Print output (drag lower right corner to resize)

```
Harry Brown
Brown
80
90
90
```

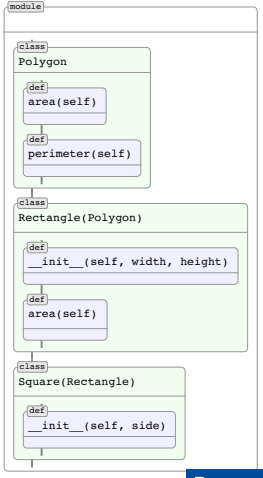


# Polymorphism and Inheritance

- **Polymorphism**: Allowing several different classes to use the same general method names
- **Inheritance**: Allowing a class to automatically reuse and extend the code of similar but more general classes

# Example 8.6 (Script and Control Flow)

```
1  class Polygon: # Creating Polygon Class
2      width = 0
3      height = 0
4      def area(self):
5          print("The area is:")
6      def perimeter(self):
7          print("The perimeter is:", 2*(self.width+self.height))
8
9  class Rectangle(Polygon): #Extending the Polygon Class
10     def __init__(self, w, h):
11         self.width = w
12         self.height = h
13     def area(self):  #Overriding the area() method
14         print("The area is:", self.width*self.height)
15
16 class Square(Rectangle): #Extending the Rectangle Class
17     def __init__(self, side):
18         Rectangle.__init__(self, side, side)
19
20 Rectangle(10,20).area()
21 Rectangle(10,20).perimeter()
22 Square(10).area()
23 Square(10).perimeter()
```





Ryerson University The Chang School of Continuing Education

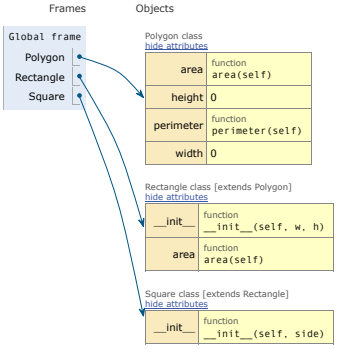
# Example 8.6 (Program Execution)

Write code in Python 3.6 (drag lower right corner to resize code editor)

```
1  class Polygon: # Creating Polygon Class
2      width = 0
3      height = 0
4      def area(self):
5          print("The area is:")
6      def perimeter(self):
7          print("The perimeter is:", 2*(self.width+self.height))
8
9  class Rectangle(Polygon):
10     def __init__(self, w, h):
11         self.width = w
12         self.height = h
13     def area(self):  #Overridding area method
14         print("The area is:", self.width*self.height)
15
16 class Square(Rectangle):
17     def __init__(self, side):
18         Rectangle.__init__(self, side, side)
19
20 Rectangle(10,20).area()
21 Rectangle(10,20).perimeter()
22 Square(10).area()
23 Square(10).perimeter()
```

Print output (drag lower right corner to resize)

```
('The area is:', 200)
('The perimeter is:', 60)
('The area is:', 100)
('The perimeter is:', 40)
```





Ryerson University The Chang School of Continuing Education

# References


- Lambert, K. A. (2010). Design with functions. In *Fundamentals of Python: From first programs through data structures* (pp. 202-218). Boston, MA: Cengage Learning
- Lambert, K. A. (2010). Design with classes. In *Fundamentals of Python: From first programs through data structures* (pp. 291-307). Boston, MA: Cengage Learning
- Python Software Foundation. (2020). Python 3.7.8 documentation. Retrieved from `https://docs.python.org/3.7/index.html`.